ФПМИ БГУ

Вычислительные методы алгебры
Лаборатоная работа 6

Решение СЛАУ итерационными методами вариационного типа

(метод нижней релаксации, метод градиентного спуска)

Подготовил:
Ткачук Павел
2 курс 1 группа

Преподаватель:
Будник Анатолий Михайлович

20 декабря 2016 г.

1 Постановка задачи

Система:

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n = f_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n}x_n = f_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n,1}x_1 + a_{n,2}x_2 + \ldots + a_{n,n}x_n = f_n \end{cases} \tag{1}$$

Входные данные:

$$\left[\begin{array}{ccccc|c} 0.6444 & 0.0000 & -0.1683 & 0.1184 & 0.1973 & 1.2677 \\ -0.0395 & 0.4208 & 0.0000 & -0.0802 & 0.0263 & 1.6819 \\ 0.0132 & -0.1184 & 0.7627 & 0.0145 & 0.0460 & -2.3657 \\ 0.0395 & 0.0000 & -0.0960 & 0.7627 & 0.0000 & -6.5369 \\ 0.0263 & -0.0395 & 0.1907 & -0.0158 & 0.5523 & 2.8351 \end{array}\right]$$

Задача:

1. Найти решение СЛАУ x итерационными методами вариационного типа(метод нижней релаксации, метод градиентного спуска) с точностью $\varepsilon = 10^{-5}$
2. Найти вектор невязки $r = Ax - f$ и количество шагов итерационнго метода

2 Алгоритм

2.1 Метод градиентного спуска

1. Приводим систему $Ax = f$ к симметричному виду $A^{(1)}x = f^{(1)}$, где
$$A^{(1)} = A^T A,\ f^{(1)} = A^T f$$
.
2. Полагаем $k = 0$, $x^{(0)} = g$ и $\epsilon = 10^{-5}$
3. Вычисляем следующее приближение
$$x^{(k+1)} = x^{(k)} - \frac{(r^{(k)}, r^{(k)})}{(Ar^{(k)}, r^{(k)})} r^{(k)},$$
$$r^{(k)} = Ax^{(k)} - f.$$
4. Если выполнено условие $\|x^{(k+1)} - x^{(k)}\| < \epsilon$, завершаем процесс и в качестве приближенного решения задачи берем $x^* \cong x^{(k+1)}$. Иначе полагаем $k = k + 1$ и переходим к пункту 3 алгоритма.

2.2 Метод нижней релаксации

1. Приводим систему $Ax = f$ к симметричному виду $A^{(1)}x = f^{(1)}$, где
$$A^{(1)} = A^T A,\ f^{(1)} = A^T f$$
.
2. Полагаем $k = 0$, $x^{(0)} = f^{(1)}$ и $\varepsilon = 10^{-5}$
3. Вычисляем следующее приближение ($\omega = 0.5$)
$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} - \omega \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{(k+1)} - \omega \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j^{(k)} + \omega \frac{f_i}{a_{ii}},\ i = 1, \ldots, n.$$
4. Если выполнено условие $\|x^{(k+1)} - x^{(k)}\| < \epsilon$, завершаем процесс и в качестве приближенного решения задачи берем $x^* \cong x^{(k+1)}$. Иначе полагаем $k = k + 1$ и переходим к пункту 3 алгоритма.

3 Результаты и вывод

3.1 Входные данные

0.6444 0.0000 -0.1683 0.1184 0.1973 1.2677
-0.0395 0.4208 0.0000 -0.0802 0.0263 1.6819
0.0132 -0.1184 0.7627 0.0145 0.0460 -2.3657
0.0395 0.0000 -0.0960 0.7627 0.0000 -6.5369
0.0263 -0.0395 0.1907 -0.0158 0.5523 2.8351

3.2 Выходные данные

```
Решаем матричное уравнение методом Гаусса-Зейделя
Основная матрица системы:
[[ 0.6444  0.     -0.1683  0.1184  0.1973]
 [-0.0395  0.4208  0.     -0.0802  0.0263]
 [ 0.0132 -0.1184  0.7627  0.0145  0.046 ]
 [ 0.0395  0.     -0.096   0.7627  0.    ]
 [ 0.0263 -0.0395  0.1907 -0.0158  0.5523]]
Свободные члены:  [ 1.2677  1.6819 -2.3657 -6.5369  2.8351]
Метод сходится
Решаем с точность 0.00001
Метод градиентного спуска
Решение  [ 0.99821875  1.99987619 -2.99975327 -9.00000936  6.00704882]
Получено за  29  итераций
Невязка  [  2.57815679e-07   4.39558916e-06   3.43799797e-06  -1.18394143e-06
  -1.69590629e-06]
Норма невязки  5.95695546733e-06
Метод нижней релаксации
Решение  [ 0.99821946  1.99986714 -2.99975844 -9.00000855  6.0070519 ]
Получено за  26  итераций
Невязка  [  2.29096549e-06   5.74128869e-07   7.26162183e-07  -4.14271435e-08
  -6.15904207e-07]
Норма невязки  2.54686327946e-06
```

3.3 Вывод

С помощью итерационных методов вариационнгго типа можно получить результат с любой точностью, однако количество шагов, требуемое для получения достаточно точного результата (в данном случае с точностью 10^{-5}), значительно меньше, чем в точных методах (Гаусса и квадратного корня).

Методу градиентного спуска на данных числах потребовалось все лишь 29 шагов для получения достаточно точного результата.

Методу нижней релаксации потребовалось 26 шагов, однако это также достаточно мало.

В целом, методы итерационного типа (при больших размерностях входных данных) выигрывают по количеству операций у точных методов, давая при этом результаты с хорошими точностями. Если сравнить количество шагов итерационных методов между собой, то наименьшее количество итераций требуется методу Гаусса-Зейделя.)

4 Листинг кода

```
import numpy as np
import numpy.linalg as linalg


def is_solvable(A, b):  # Проверка сходимости
    size = len(A)
    E = np.eye(size)
    B = np.array(E - np.dot(A, np.transpose(A)) / linalg.norm(np.dot(A, np.transpose(A))))
    sums = []
    for i in range(size):
        sums.append(sum(abs(B[i, j]) for j in range(size)))
    return max(sums) < 1


def bottom_relax(A, b, eps):  # Нижняя релаксация
    size = len(A)
    count = 0
    converge = False
    x = b
    w = 0.5
    while not converge:
        count += 1
        x_new = x.copy()
        for i in range(size):
            s1 = sum(A[i][j] * x_new[j] for j in range(i))
            s2 = sum(A[i][j] * x[j] for j in range(i + 1, size))
            x_new[i] = (1 - w) * x[i] + (w * (b[i] - s1 - s2) / A[i][i])
        converge = linalg.norm(x - x_new) <= w * eps
        x = x_new
    return x, count


def grad_descent(A, b, eps):  # Градиентный спуск
    b = np.dot(np.transpose(A), b)
    A = np.dot(np.transpose(A), A)
    x = b
    r = np.dot(A, x) - b
    count = 0
    converge = False
    while not converge:
        count += 1
        r = np.dot(A, x) - b
        x_new = x - np.dot(r, r) * r / np.dot(np.dot(A, r), r)
        converge = linalg.norm(x - x_new) <= eps
        x = x_new
    return x, count


file = open("matrix", "r")  # Чтение файла
A, b = [], []
for line in file:
    A.append([float(el) for el in line.split()[:-1]])
    b.append(float(line.split().pop()))
A = np.array(A)
b = np.array(b)
print("Решаем матричное уравнение методом Гаусса-Зейделя")
```

```
print("Основная матрица системы:")
print(A)
print("Свободные члены: ", b)
if is_solvable(A, b):
    print("Метод сходится")
    print("Решаем с точность 0.00001")
    print("Метод градиентного спуска")
    ans = grad_descent(A, b, 0.00001)
    print("Решение ", ans[0], "\nПолучено за ", ans[1], " итераций")
    print("Невязка ", np.dot(A, ans[0]) - b)
    print("Норма невязки ", linalg.norm(np.dot(A, ans[0]) - b))

    print("Метод нижней релаксации")
    ans = bottom_relax(A, b, 0.00001)
    print("Решение ", ans[0], "\nПолучено за ", ans[1], " итераций")
    print("Невязка ", np.dot(A, ans[0]) - b)
    print("Норма невязки ", linalg.norm(np.dot(A, ans[0]) - b))
else:
    print("Метод расходится")
```